ФТФ, 2-ый курс, ИВТ-Б(2), Королёв Алексей.

**Ссылка на GitHub:** <https://github.com/ShEIH24/Lab1_Py>

**ЛАБОРАТОРНАЯ РАБОТА №2**

**Тема:** Работа с файловой системой и процессами в Python.

**Цель:** Научиться программировать на языке Python скрипты, взаимодействующие с файлами и процессами операционной системы.

**Индивидуальные задания:**

1. Напишите скрипт, который читает текстовый файл и выводит символы в порядке убывания частоты встречаемости в тексте. Регистр символа не имеет значения. Программа должна учитывать только буквенные символы (символы пунктуации, цифры и служебные символы следует игнорировать). Проверьте работу скрипта на нескольких файлах с текстом на английском и русском языках, сравните результаты с таблицами, приведенными в wikipedia.org/wiki/Letter\_frequencies.
2. Напишите скрипт, позволяющий искать в заданной директории и в ее подпапках файлы-дубликаты на основе сравнения контрольных сумм (MD5). Файлы могут иметь одинаковое содержимое, но отличаться именами. Скрипт должен вывести группы имен обнаруженных файлов дубликатов.
3. Задан путь к директории с музыкальными файлами (в названии которых нет номеров, а только названия песен) и текстовый файл, хранящий полный список песен с номерами и названиями в виде строк формата «01. Freefall [6:12]». Напишите скрипт, который корректирует имена файлов в директории на основе текста списка песен.
4. Напишите скрипт, который позволяет ввести с клавиатуры имя текстового файла, найти в нем с помощью регулярных выражений все подстроки определенного вида, в соответствии с вариантом. Например, для варианта № 1 скрипт должен вывести на экран следующее:

Строка 3, позиция 10 : найдено '11-05-2014'

Строка 12, позиция 2 : найдено '23-11-2014'

Строка 12, позиция 17 : найдено '23-11-2014'

*Вариант 5: найдите все номера телефонов – подстроки вида «(000)1112233» или «(000)111-22-33»*

1. Введите с клавиатуры текст. Программно найдите в нем и выведите отдельно все слова, которые начинаются с большого латинского символа (от A до Z) и заканчиваются 2 или 4 цифрами, например *«Petr93», «Johnny70», «Service2002»*. Используйте регулярные выражения.
2. Напишите скрипт reorganize.py, который в директории --source создает две директории: Archive и Small. В первую директорию помещаются файлы с датой изменения, отличающейся от текущей даты на количество дней более параметра --days (т.е. относительно старые файлы). Во вторую – все файлы размером меньше параметра --size байт. Каждая директория должна создаваться только в случае, если найден хотя бы один файл, который должен быть в нее помещен. Пример вызова:

reorganize --source "C:\TestDir" --days 2 --size 4096

1. Написать скрипт trackmix.py, который формирует обзорный трек-микс альбома (попурри из коротких фрагментов mp3-файлов в пользовательской директории). Для манипуляций со звуковыми файлами можно использовать сторонние утилиты, например, FFmpeg. Пример вызова и работы скрипта:

trackmix --source "C:\Muz\Album" --count 5 --frame 15 -l -e

--- processing file 1: 01 - Intro.mp3

--- processing file 2: 02 - Outro.mp3

--- done!

Параметры скрипта:

--source (-s) – имя рабочей директории с треками, обязателен;

--destination (-d) – имя выходного файла, необязателен (если не указан, то имя выходного файла – mix.mp3 в директории source);

--count (-c) – количество файлов в "нарезке", необязателен (если он не указан, то перебираются все mp3-файлы в директории source);

--frame (-f) – количество секунд на каждый файл, необязателен (если не указан, скрипт вырезает по 10 секунд из произвольного участка каждого файла);

--log (-l) – необязательный ключ (если этот ключ указывается, то скрипт должен выводить на консоль лог процесса обработки файлов, как в примере);

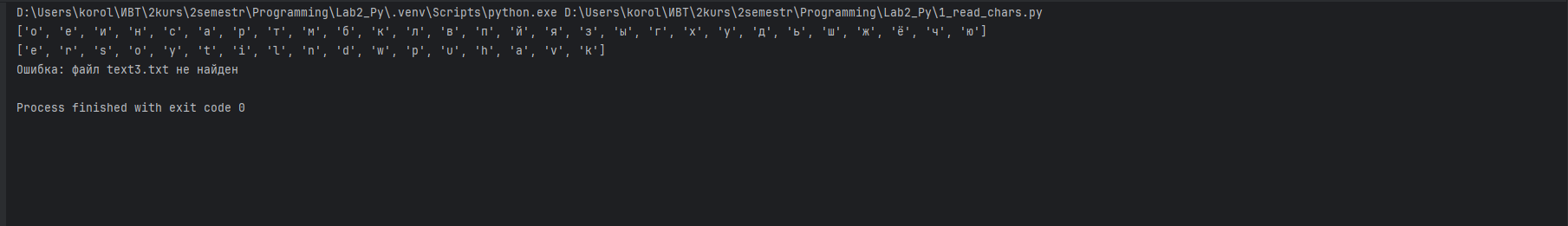
--extended (-e) – необязательный ключ (если этот ключ указывается, то каждый фрагмент попурри начинается и завершается небольшим fade in/fade out).

**Ход работы:**

1. Код программы:

def read\_chars(filename):  
 char\_count = {}  
  
 try:  
 with open(filename, encoding='utf-8') as f:  
 for line in f:  
 for char in line.lower():  
 if char.isalpha():  
 char\_count[char] = char\_count.get(char, 0) + 1  
  
 # Сортируем символы по частоте (по убыванию)  
 sorted\_chars = sorted(char\_count.keys(),  
 key=lambda x: char\_count[x],  
 reverse=True)  
  
 return sorted\_chars  
  
 except FileNotFoundError:  
 return f"Ошибка: файл {filename} не найден"  
 except Exception as e:  
 return f"Произошла ошибка при чтении файла: {str(e)}"  
  
print(read\_chars('text1.txt'))  
print(read\_chars('text2.txt'))  
print(read\_chars('text3.txt'))

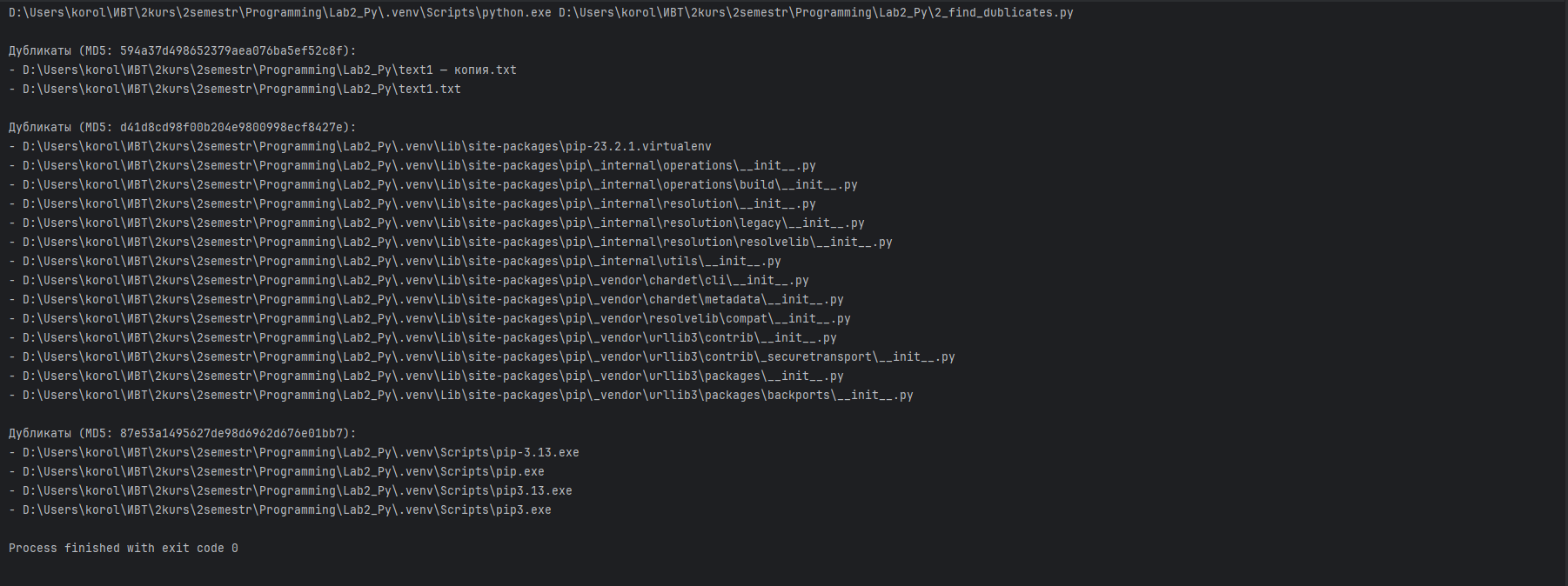
Вывод программы:



1. Код программы:

import os  
import hashlib  
from collections import defaultdict  
  
  
def get\_file\_hash(filepath):  
 *"""Вычисляет MD5-хеш файла"""* md5\_hash = hashlib.md5()  
  
 with open(filepath, 'rb') as f:  
 # Читаем файл блоками по 8192 байта  
 while chunk := f.read(8192):  
 md5\_hash.update(chunk)  
  
 return md5\_hash.hexdigest()  
  
  
def find\_duplicates(directory):  
 *"""  
 Находит дубликаты файлов в директории и подпапках.  
 Возвращает словарь, где ключ - хеш, значение - список путей к файлам  
 """* hash\_dict = defaultdict(list)  
  
 # Обходим все файлы в директории и подпапках  
 for root, \_, files in os.walk(directory):  
 for filename in files:  
 filepath = os.path.join(root, filename)  
 try:  
 file\_hash = get\_file\_hash(filepath)  
 hash\_dict[file\_hash].append(filepath)  
 except:  
 continue  
  
 # Оставляем только группы с дубликатами (больше одного файла)  
 return {hash\_val: paths for hash\_val, paths in hash\_dict.items()  
 if len(paths) > 1}  
  
  
def print\_duplicates(duplicates):  
 *"""Выводит найденные группы дубликатов"""* if not duplicates:  
 print("Дубликаты не найдены")  
 return  
  
 for hash\_val, file\_list in duplicates.items():  
 print(f"\nДубликаты (MD5: {hash\_val}):")  
 for file\_path in file\_list:  
 print(f"- {file\_path}")  
  
directory = r'D:\Users\korol\ИВТ\2kurs\2semestr\Programming\Lab2\_Py'  
dupes = find\_duplicates(directory)  
print\_duplicates(dupes)

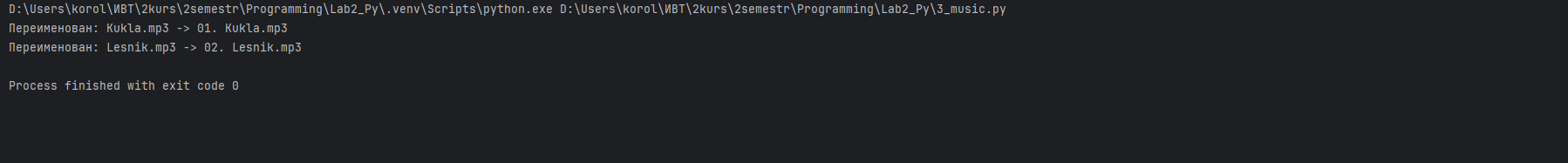
Вывод программы:



1. Код программы:

import os  
import re  
  
  
def rename\_music\_files(music\_dir, tracklist\_path):  
 *"""  
 Переименовывает музыкальные файлы в директории согласно списку треков из текстового файла.  
  
 Args:  
 music\_dir (str): Путь к директории с музыкальными файлами  
 tracklist\_path (str): Путь к текстовому файлу со списком треков  
 """* # Читаем файл со списком треков и создаем словарь {название: номер}  
 tracks = {}  
 with open(tracklist\_path, 'r', encoding='utf-8') as f:  
 for line in f:  
 # Ищем строки формата "01. Freefall [6:12]"  
 if match := re.match(r'(\d+)\.\s+(.+?)\s\*\[\d+:\d+\]', line.strip()):  
 tracks[match.group(2).strip()] = match.group(1)  
  
 # Получаем список музыкальных файлов из директории  
 music\_files = [f for f in os.listdir(music\_dir)  
 if f.lower().endswith(('.mp3', '.flac', '.m4a', '.wav'))]  
  
 # Для каждого трека из списка ищем и переименовываем файл  
 for song\_name, track\_number in tracks.items():  
 # Ищем файл с соответствующим названием (без учета регистра)  
 matching\_file = next(  
 (f for f in music\_files  
 if os.path.splitext(f)[0].lower() == song\_name.lower()),  
 None  
 )  
  
 if matching\_file:  
 # Формируем новое имя, сохраняя оригинальное расширение  
 new\_name = f"{track\_number}. {song\_name}{os.path.splitext(matching\_file)[1]}"  
 try:  
 # Переименовываем файл  
 os.rename(  
 os.path.join(music\_dir, matching\_file),  
 os.path.join(music\_dir, new\_name)  
 )  
 print(f"Переименован: {matching\_file} -> {new\_name}")  
 except OSError as e:  
 print(f"Ошибка при переименовании {matching\_file}: {e}")  
 else:  
 print(f"Не найден файл для трека: {song\_name}")  
  
rename\_music\_files(r'D:\Users\korol\ИВТ\2kurs\2semestr\Programming\Lab2\_Py\music\_example', r'D:\Users\korol\ИВТ\2kurs\2semestr\Programming\Lab2\_Py\music\_example\names.txt')

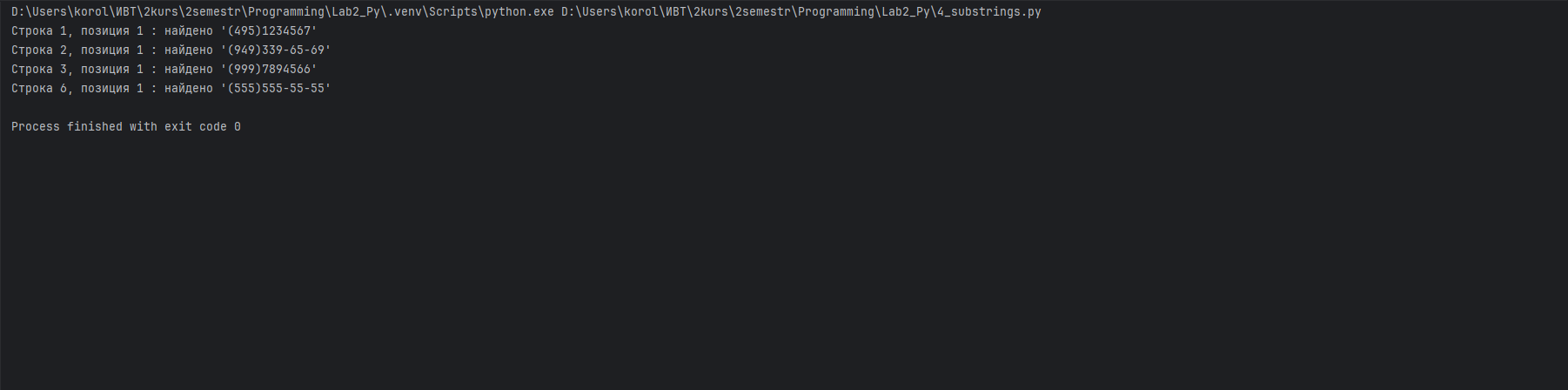
Вывод программы:



1. Код программы:

import re  
  
  
def find\_numbers(file):  
 pattern = r'\(\d{3}\)(?:\d{7}|\d{3}-\d{2}-\d{2})'  
 try:  
 with open(file, 'r', encoding='utf-8') as f:  
 # Читаем файл построчно  
 for line\_num, line in enumerate(f, 1):  
 # Ищем все совпадения в текущей строке  
 for match in re.finditer(pattern, line):  
 print(f"Строка {line\_num}, позиция {match.start() + 1} : найдено '{match.group()}'")  
  
 except FileNotFoundError:  
 print(f"Файл {file} не найден")  
 except Exception as e:  
 print(f"Произошла ошибка: {e}")  
  
  
find\_numbers('numbers.txt')

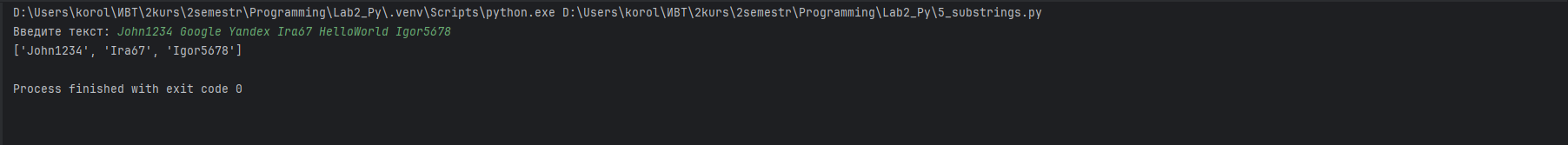
Вывод программы:



1. Код программы:

import re  
  
def find\_word (text):  
 return list(map(lambda x: x[0] + x[1], re.findall(r'(\b[A-Z][a-z]+\d{2}\b)|(\b[A-Z][a-z]+\d{4}\b)', text)))  
  
  
message = input('Введите текст: ')  
print(find\_word(message))

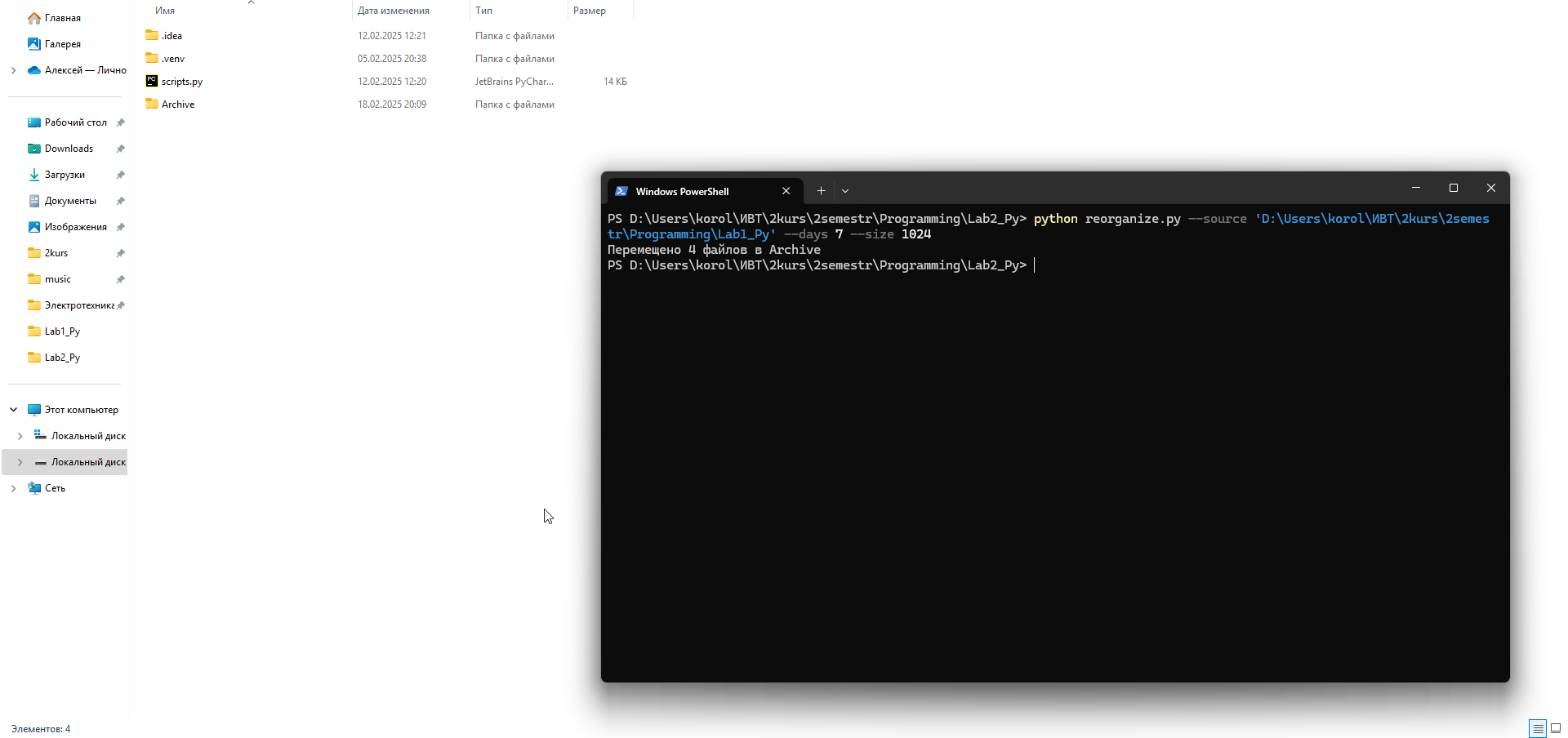
Вывод программы:



1. Код программы:

import os  
import shutil  
import argparse  
from datetime import datetime  
  
  
def reorganize(source, days, size):  
 *"""  
 Реорганизует файлы в указанной директории:  
 - перемещает старые файлы в папку Archive  
 - перемещает маленькие файлы в папку Small  
 """* # Проверяем существование исходной директории  
 if not os.path.exists(source):  
 print(f"Ошибка: Директория '{source}' не существует")  
 return  
  
 # Списки для хранения файлов  
 archive\_files = []  
 small\_files = []  
  
 # Текущая дата для сравнения  
 current\_date = datetime.now()  
  
 # Проверяем все файлы в директории  
 for filename in os.listdir(source):  
 path = os.path.join(source, filename)  
  
 # Пропускаем папки  
 if os.path.isdir(path):  
 continue  
  
 # Получаем информацию о файле  
 stats = os.stat(path)  
 days\_old = (current\_date - datetime.fromtimestamp(stats.st\_mtime)).days  
  
 # Проверяем условия и добавляем файлы в соответствующие списки  
 if days\_old > days:  
 archive\_files.append(filename)  
 if stats.st\_size < size:  
 small\_files.append(filename)  
  
 # Обрабатываем старые файлы  
 if archive\_files:  
 archive\_dir = os.path.join(source, 'Archive')  
 os.makedirs(archive\_dir, exist\_ok=True)  
 for file in archive\_files:  
 shutil.move(os.path.join(source, file),  
 os.path.join(archive\_dir, file))  
 print(f"Перемещено {len(archive\_files)} файлов в Archive")  
  
 # Обрабатываем маленькие файлы  
 small\_files = [f for f in small\_files if f not in archive\_files] # Исключаем уже перемещенные  
 if small\_files:  
 small\_dir = os.path.join(source, 'Small')  
 os.makedirs(small\_dir, exist\_ok=True)  
 for file in small\_files:  
 shutil.move(os.path.join(source, file),  
 os.path.join(small\_dir, file))  
 print(f"Перемещено {len(small\_files)} файлов в Small")  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Настраиваем параметры командной строки  
 parser = argparse.ArgumentParser(description='Сортировка файлов по размеру и дате изменения')  
 parser.add\_argument('--source', required=True, help='Путь к директории')  
 parser.add\_argument('--days', type=int, required=True, help='Пороговое значение дней')  
 parser.add\_argument('--size', type=int, required=True, help='Пороговое значение размера в байтах')  
  
 # Получаем аргументы и запускаем основную функцию  
 args = parser.parse\_args()  
 reorganize(args.source, args.days, args.size)

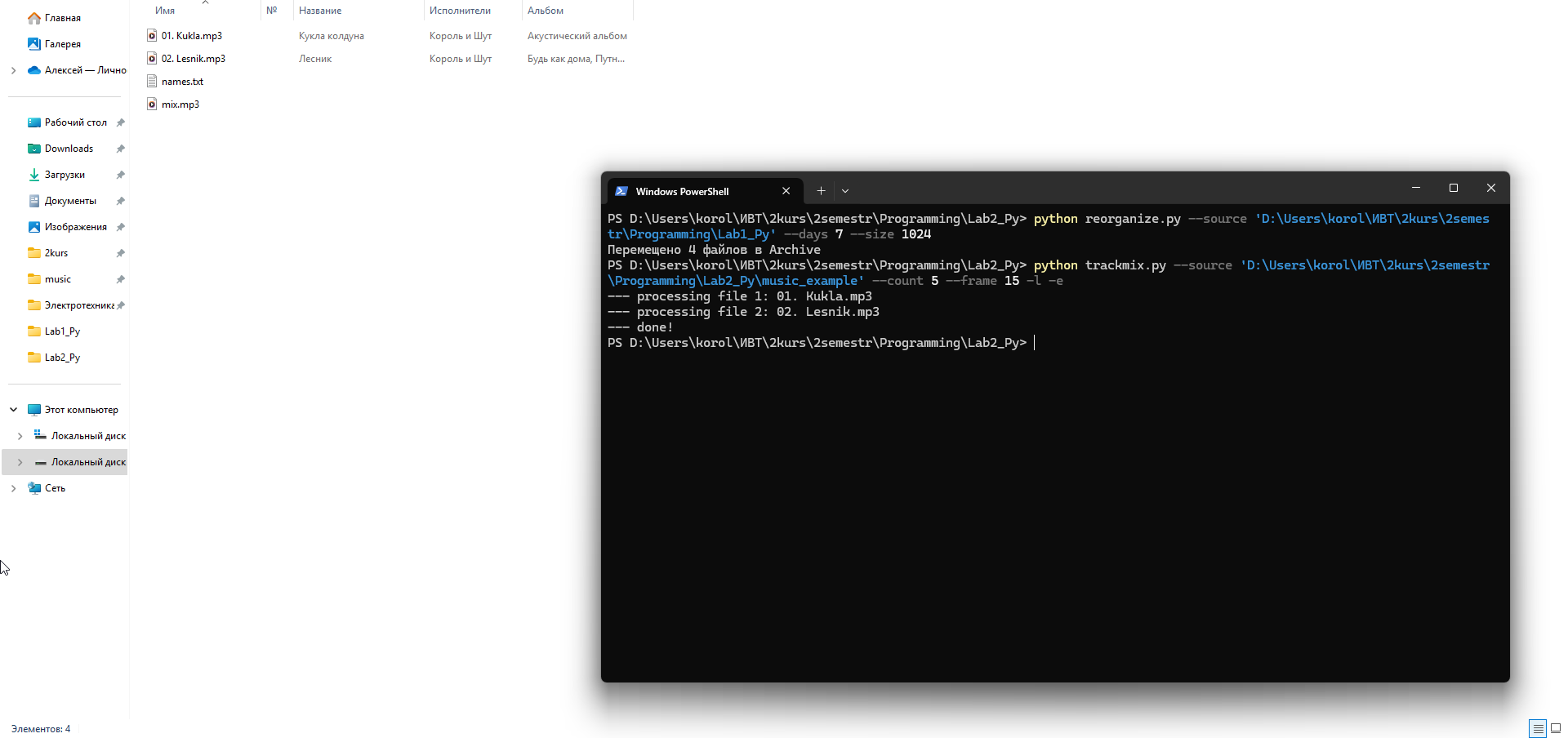
Вывод программы:



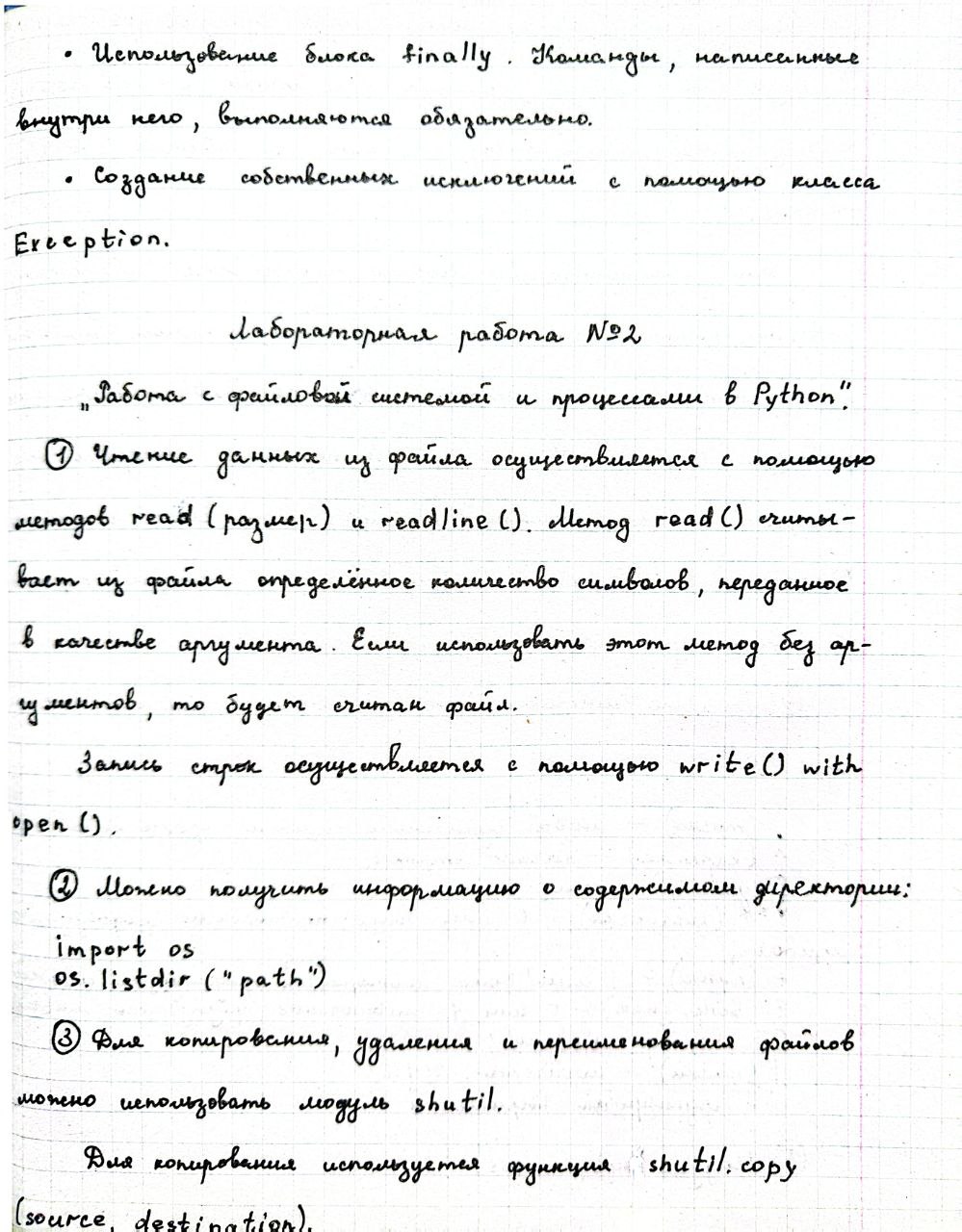
1. Код программы:

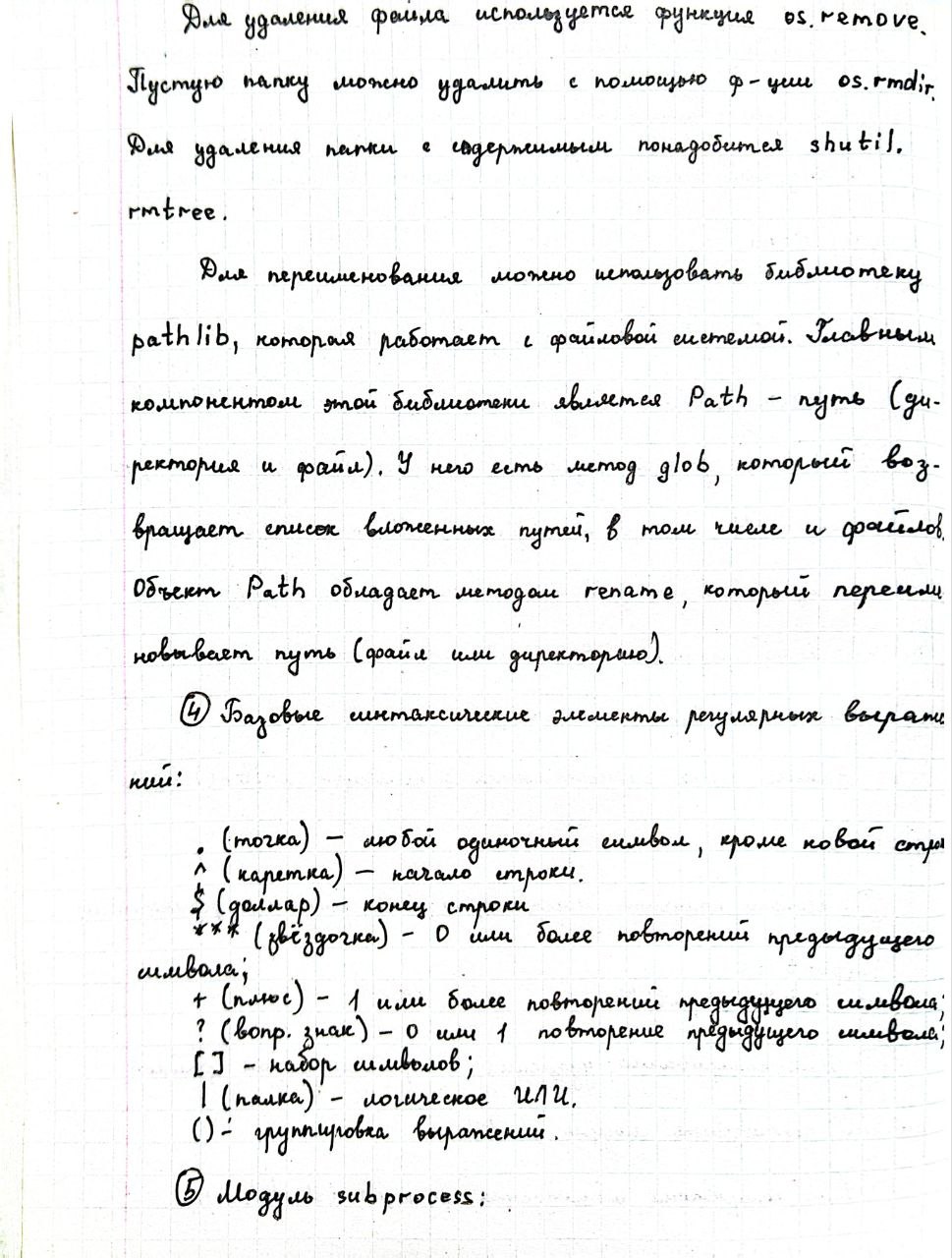
import os  
import argparse  
import subprocess  
import random  
import sys  
from glob import glob  
  
  
def check\_ffmpeg():  
 *"""Проверяет, установлен ли FFmpeg в системе"""* try:  
 subprocess.run(['ffmpeg', '-version'], capture\_output=True)  
 return True  
 except FileNotFoundError:  
 print("Ошибка: FFmpeg не установлен или не найден в PATH")  
 print("Установите FFmpeg и добавьте его в PATH")  
 return False  
  
  
def trackmix(source, destination=None, count=None, frame=10, log=False, extended=False):  
 *"""  
 Создает музыкальный микс из фрагментов MP3 файлов  
  
 Параметры:  
 source (str): Путь к директории с MP3 файлами  
 destination (str): Путь к выходному файлу (по умолчанию mix.mp3 в source)  
 count (int): Количество файлов для обработки (по умолчанию все MP3)  
 frame (int): Длительность фрагмента в секундах (по умолчанию 10)  
 log (bool): Выводить ли лог обработки  
 extended (bool): Применять ли fade эффекты  
 """* # Проверяем наличие FFmpeg  
 if not check\_ffmpeg():  
 return  
  
 # Проверяем существование исходной директории  
 if not os.path.exists(source):  
 print(f"Ошибка: Директория '{source}' не существует")  
 return  
  
 # Если выходной файл не указан, создаем его в исходной директории  
 if destination is None:  
 destination = os.path.join(source, "mix.mp3")  
  
 # Получаем список всех MP3 файлов в директории  
 mp3\_files = glob(os.path.join(source, "\*.mp3"))  
  
 if not mp3\_files:  
 print(f"Ошибка: В директории '{source}' не найдены MP3 файлы")  
 return  
  
 # Если указано количество файлов, берем только нужное количество  
 if count is not None:  
 count = min(count, len(mp3\_files))  
 mp3\_files = mp3\_files[:count]  
  
 # Временная директория для фрагментов  
 temp\_dir = os.path.join(source, "temp")  
 os.makedirs(temp\_dir, exist\_ok=True)  
  
 try:  
 # Список временных файлов  
 temp\_files = []  
  
 # Обрабатываем каждый файл  
 for i, mp3\_file in enumerate(mp3\_files, 1):  
 if log:  
 print(f"--- processing file {i}: {os.path.basename(mp3\_file)}")  
  
 # Получаем длительность файла  
 try:  
 duration = float(subprocess.check\_output([  
 'ffprobe', '-v', 'quiet',  
 '-show\_entries', 'format=duration',  
 '-of', 'default=noprint\_wrappers=1:nokey=1',  
 mp3\_file  
 ]))  
 except subprocess.CalledProcessError:  
 print(f"Ошибка при чтении файла: {mp3\_file}")  
 continue  
  
 # Выбираем случайную позицию  
 start = random.uniform(0, max(0, duration - frame))  
  
 # Временный файл для фрагмента  
 temp\_file = os.path.join(temp\_dir, f"temp\_{i}.mp3")  
  
 # Формируем команду FFmpeg  
 ffmpeg\_cmd = ['ffmpeg', '-y', '-i', mp3\_file,  
 '-ss', str(start), '-t', str(frame)]  
  
 # Добавляем fade эффекты если нужно  
 if extended:  
 ffmpeg\_cmd.extend([  
 '-af', f'afade=t=in:st=0:d=1,afade=t=out:st={frame - 1}:d=1'  
 ])  
  
 ffmpeg\_cmd.append(temp\_file)  
  
 # Выполняем команду  
 try:  
 subprocess.run(ffmpeg\_cmd, capture\_output=True, check=True)  
 temp\_files.append(temp\_file)  
 except subprocess.CalledProcessError:  
 print(f"Ошибка при обработке файла: {mp3\_file}")  
 continue  
  
 if not temp\_files:  
 print("Ошибка: Не удалось создать фрагменты")  
 return  
  
 # Создаем файл со списком для объединения  
 concat\_file = os.path.join(temp\_dir, "files.txt")  
 with open(concat\_file, 'w', encoding='utf-8') as f:  
 for temp\_file in temp\_files:  
 f.write(f"file '{os.path.basename(temp\_file)}'\n")  
  
 # Объединяем все фрагменты  
 try:  
 subprocess.run([  
 'ffmpeg', '-y',  
 '-f', 'concat',  
 '-safe', '0',  
 '-i', concat\_file,  
 '-c', 'copy',  
 destination  
 ], capture\_output=True, check=True)  
 except subprocess.CalledProcessError:  
 print("Ошибка при создании финального микса")  
 return  
  
 if log:  
 print("--- done!")  
  
 finally:  
 # Удаляем временные файлы  
 for temp\_file in temp\_files:  
 if os.path.exists(temp\_file):  
 os.remove(temp\_file)  
 if os.path.exists(concat\_file):  
 os.remove(concat\_file)  
 if os.path.exists(temp\_dir):  
 os.rmdir(temp\_dir)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 parser = argparse.ArgumentParser(description='Создание музыкального микса из MP3 файлов')  
 parser.add\_argument('--source', '-s', required=True,  
 help='Директория с MP3 файлами')  
 parser.add\_argument('--destination', '-d',  
 help='Путь к выходному файлу (по умолчанию mix.mp3)')  
 parser.add\_argument('--count', '-c', type=int,  
 help='Количество файлов для обработки')  
 parser.add\_argument('--frame', '-f', type=int, default=10,  
 help='Длительность фрагмента в секундах (по умолчанию 10)')  
 parser.add\_argument('--log', '-l', action='store\_true',  
 help='Выводить лог обработки')  
 parser.add\_argument('--extended', '-e', action='store\_true',  
 help='Применять fade эффекты')  
  
 args = parser.parse\_args()  
 trackmix(  
 args.source,  
 args.destination,  
 args.count,  
 args.frame,  
 args.log,  
 args.extended  
 )

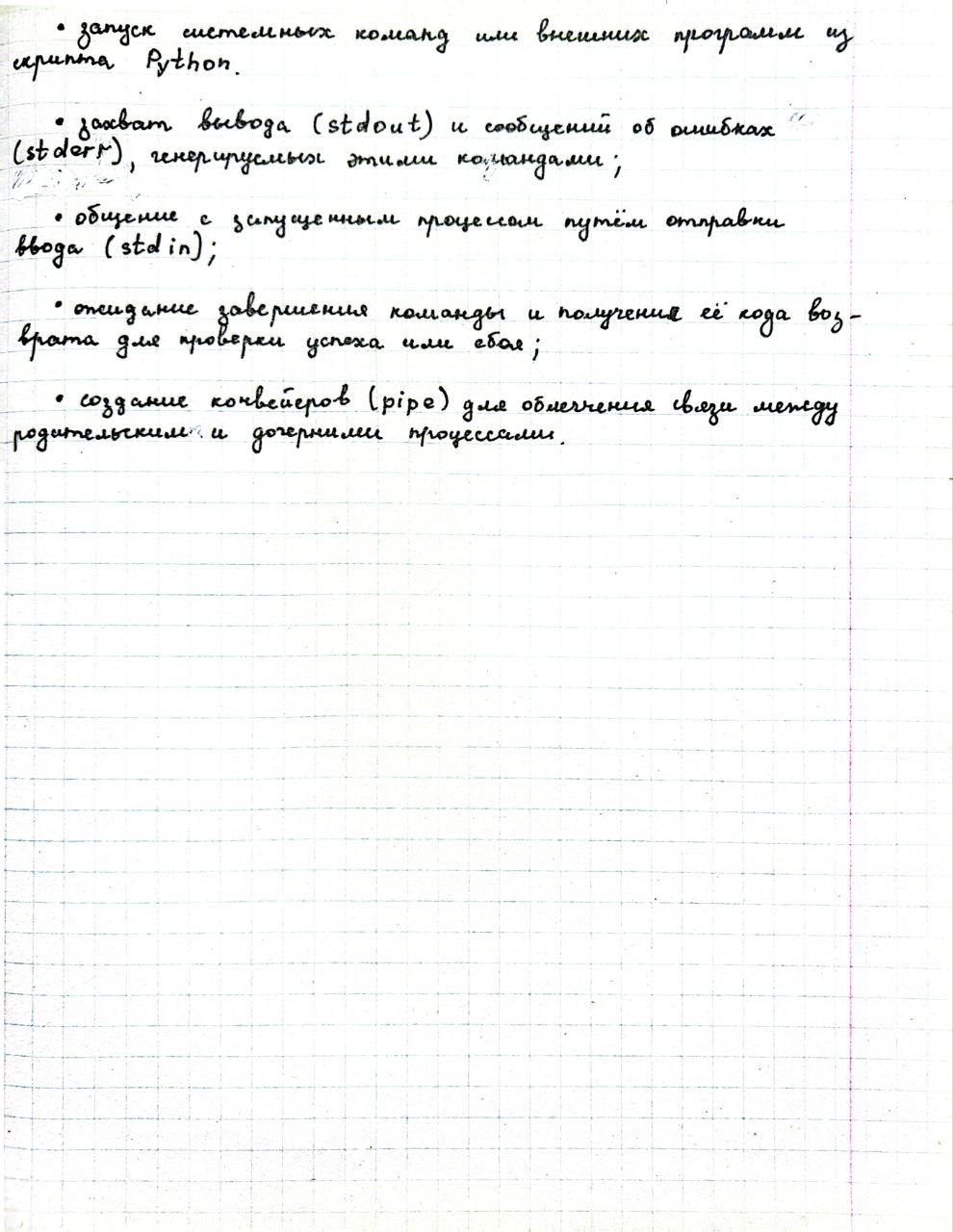
Вывод программы:



Контрольные вопросы







Вывод: Научился программировать на языке Python скрипты, взаимодействующие с файлами и процессами операционной системы.